REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 1992	Technical Memorandum

4. TITLE AND SUBTITLE
Graphical Workstation Capability for Reliability Modeling

5. FUNDING NUMBERS
WU 505-66-21

6. AUTHOR(S)
Salvatore J. Bavuso, Sandra V. Koppen, and Pamela J. Haley

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, VA 23665-5225

8. PERFORMING ORGANIZATION REPORT NUMBER
L-16887

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA TM-4317

11. SUPPLEMENTARY NOTES
Bavuso and Haley: Langley Research Center, Hampton, VA; Koppen: Lockheed Engineering & Sciences Company, Hampton, VA.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified–Unlimited

Subject Category 61

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
In addition to computational capabilities, software tools for estimating the reliability of fault-tolerant digital computer systems must also provide a means of interfacing with the user. This paper describes the new graphical interface capability of the hybrid automated reliability predictor (HARP), a software package that implements advanced reliability modeling techniques. The graphics oriented (GO) module provides the user with a graphical language for modeling system failure modes through the selection of various fault-tree gates, including sequence-dependency gates, or by a Markov chain. By using this graphical input language, a fault tree becomes a convenient notation for describing a system. In accounting for any sequence dependencies, HARP converts the fault-tree notation to a complex stochastic process that is reduced to a Markov chain, which it can then solve for system reliability. The graphics capability is available for use on an IBM-compatible PC, a Sun, and a VAX workstation. The GO module is written in the C programming language and uses the graphical kernel system (GKS) standard for graphics implementation. The PC, VAX, and Sun versions of the HARP GO module are currently in beta-testing stages.

14. SUBJECT TERMS	15. NUMBER OF PAGES
HARP; Reliability; Graphics; Fault tree; PC; Markov; Workstation; Fault tolerance	11
	16. PRICE CODE A03

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified		

Abstract

In addition to computational capabilities, software tools for estimating the reliability of fault-tolerant digital computer systems must also provide a means of interfacing with the user. This paper describes the new graphical interface capability of the hybrid automated reliability predictor (HARP), a software package that implements advanced reliability modeling techniques. The graphics oriented (GO) module provides the user with a graphical language for modeling system failure modes through the selection of various fault-tree gates, including sequence-dependency gates, or by a Markov chain. By using this graphical input language, a fault tree becomes a convenient notation for describing a system. In accounting for any sequence dependencies, HARP converts the fault-tree notation to a complex stochastic process that is reduced to a Markov chain, which it can then solve for system reliability. The graphics capability is available for use on an IBM-compatible PC, a Sun, and a VAX workstation. The GO module is written in the C programming language and uses the graphical kernel system (GKS) standard for graphics implementation. The PC, VAX, and Sun versions of the HARP GO module are currently in beta-testing stages.

Introduction

The specification of a reliability model can take a number of different forms. Some of the more popular notations include: reliability block diagrams, fault trees, Markov chains, and Petri nets. The popularity of a particular form is often related to the application and background of the modeler; however, models resulting from each representation overlap to varying degrees. Typically, a notation is chosen because of the user's familiarity with it. Modeling, the process of abstracting reality, is an art, and the notation is the confident expression of that reality.

Unfortunately, a particular notation is rarely robust enough to map into every reliability model of interest. It is becoming increasingly necessary for the reliability analyst to be fluent with more than one notation. This requirement has come about because new system architectures that use embedded computers are becoming increasingly more complex to model in both size and dynamics. The modeling problem is exacerbated by the necessity to arrive at a numerical solution. A synergistic marriage of the two capabilities is essential. There are many models for which a practical numerical solution is unattainable.

For a large class of system designs, the combination of fault-tree or reliability block diagram models and Monte Carlo simulation is feasible. For many new system designs that include computers, particularly of the fault-tolerant variety, classical Monte Carlo simulation becomes impractical because of the amount of solution time required.

Attempts to bridge the gap between the desire to include the dynamics of fault handling and the desire to retain the familiar and popular fault-tree notation have been moderately successful in recent years by combining the succinct fault-tree notation with the modeling dynamics of the Markov chain. One successful attempt (ref. 1) was the hybrid automated reliability predictor (HARP), which allows the user to describe the fault-occurrence model as a dynamic fault tree by using a notation that includes sequence-dependency gates. What is unique about HARP is that it converts the dynamic fault-tree model notation into a Markov chain and solves a Markov chain by using a standard, well-known, numerical-integration algorithm. Until now, the only possible, but impractical, method to arrive at a solution to such a dynamic fault-tree model was the solution by Monte Carlo simulation. For many system designs, even those that require huge Markov models, HARP arrives at solutions that are affordable.

Another important aspect of a modeling notation is its presentation. It is a common practice to make a pictorial representation of a reliability model prior to solution. The sketch and numerical data are transcribed into a textual notation for input to a computer program. For many practical applications, this translation may be error prone.

With the advent of personal computers and computer workstations, a graphical notational input is becoming a practical necessity. Aside from the necessity for affordable graphical computational capability, the realization of such a graphical notational input rests heavily on the presence of a standard graphical language to insure widespread portability. Although several graphical standards existed for years, large support for any one standard was absent until IBM announced graphical kernel system (GKS) support for their PC products (ref. 2). This announcement prompted Langley Research Center and the developers of HARP at Duke University to embark on the development of a graphical input notation for HARP.

Overview of HARP

HARP is a software tool for analytically predicting the reliability of fault-tolerant digital computer systems; it is also applicable to a very large class of systems in general. In addition to reliability, it

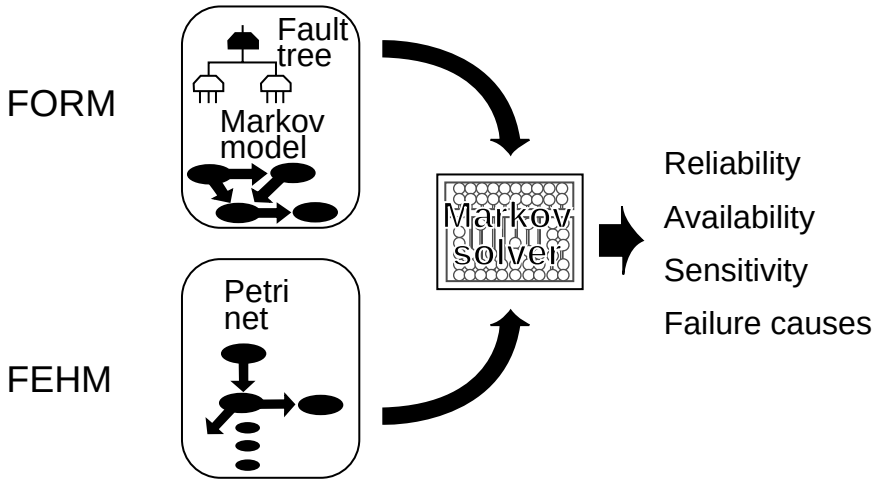

Figure 1. FORM and FEHM submodels.

can be used to analyze system availability, sensitivity, and failure causes. Its notable features include very large system modeling, dynamic fault modeling, automatic conversion of fault-tree input to a Markov chain or manual Markov chain input, automatic insertion of fault-handling models into Markov chains, automatic parametric analysis, and portability of the code. HARP utilizes a method called behavioral decomposition to solve for the reliability of a system when fault/error handling is modeled. A discussion on this subject follows; however, the reader should see references 1 and 3 for more details.

When fault/error handling is considered, dependencies exist between stochastic events that make it necessary and practical to use a Markovian representation of the reliability model. A Markov process contains information about a system's fault processes, component depletion, and recovery procedures. Graphically, a Markov model consists of states and transitions. The states contain information about the number of operational components, and the transitions are rates at which specific components fail, which causes a change in the state of the system. Computations are done to determine the probability of being in a state based on time. The reliability of the system can then be determined by adding the probabilities of the operational states (ref. 1). However, in systems designed with fault tolerance, a very large model state space, which introduces computational problems, can result. These problems can be solved by utilizing the methods of decomposition and aggregation, that is, dividing the system into smaller subsystems based on component types, solving these subsystems separately, and then combining the results of the subsystems to produce the solution of the larger system. However, this method requires that the behaviors of the subsystems be independent. In many fault-tolerant systems, this is a false assumption, because these systems may include dependencies.

HARP offers an alternative approach called behavioral decomposition (ref. 4). Using this method, HARP allows a user to segregate a reliability model into two submodels, a fault-occurrence/repair model (FORM) and fault/error-handling model (FEHM). The FORM describes a system as a fault tree or a Markov chain and relates information about hardware redundancy and fault processes. Using the FEHM to describe specific recovery procedures, a user can include details about permanent, transient, and intermittent faults in a reliability model. Figure 1 illustrates the behavioral decomposition method with FORM and FEHM submodels. HARP provides a user with seven FEHM's, which range from simple (probabilities and moments) to very complex (an extended stochastic Petri net). The model can be input into HARP by using an interactive textually oriented interface or a graphically oriented interface. If the FORM is a fault tree, it is first converted to a complex stochastic process that is reduced to a Markov chain. The FEHM's are solved separately from the FORM to determine the exit probabilities and holding times for transient restoration (R), permanent coverage (C), near-coincident fault failures (N), and single-point failures (S).

Figure 2 demonstrates the basic concept of how HARP inserts FEHM's into a Markov chain. Circles represent operational states of the system, and octagons represent system failure states. The "32" in the upper left circle indicates that this

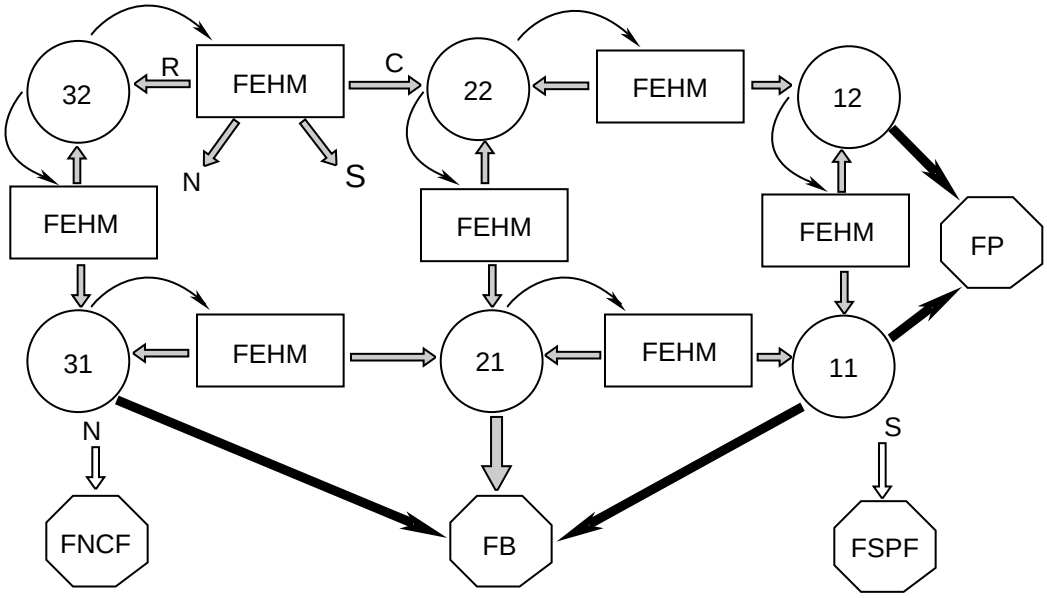

Figure 2. Representation of combined FORM/FEHM model Markov chain.

state represents the system condition in which three processor subsystems and two bus subsystems are operational. The Markov process enters the failure processor (FP) failure state when all system processors have failed and enters the failure bus (FB) failure state when all bus subsystems have failed. System failure near-coincident failure (FNCF) and failure single-point failure (FSPF) failures states are entered when a near-coincident or single-point system failure occurs. The sum of the probabilities of entering the octagon states is the system failure probability, the metric of most interest to reliability engineers. The FEHM's are shown as rectangles and are automatically inserted between operational states by HARP. The Markov process moves from left to right when processor failures occur and from top to bottom for bus failures; therefore, FEHM's for processors are inserted left to right and bus FEHM's are inserted top to bottom. The FEHM exit probabilities are used to alter the transition rates between the states in the Markov chain FORM, which is not depicted here. No matter how complex the FEHM models may be, and no matter how many FEHM's are specified, this process produces at most two additional system failure states in the chain, which represent near-coincident fault failures and single-point failures. (The reduction of an enormous number of Markov states for most practical systems is the forte of behavioral decomposition.) The model is then given to a common ordinary differential-equation solver to compute the results.

HARP is composed of three modules that are written in the Fortran programming language. The first module, TDRIVE, executes a textual user interface and converts a fault tree to a Markov chain. The second HARP module, FIFACE, builds the symbolic transition rate matrix, and the third HARP module, HARPENG, performs the computation.

HARP applications include aircraft life-critical systems, civilian-aircraft electronics, military avionics, space systems, and nuclear power control systems. References 3 and 5 contain more details on specific systems and architectures for which HARP has been applied.

HARP was developed on a Sun 3 computing platform running under Berkeley Unix 4.3. The source code was written in ANSI standard Fortran 77. HARP has been ported to a large host of computing platforms; the major operating systems are DEC VMS and Ultrix, Berkeley Unix 4.3, AT&T Unix 5.2, and MS-DOS. PC-HARP running under MS-DOS is a scaled-down version of HARP that executes on IBM-compatible 286/386/486 machines. Certain limitations are placed on PC-HARP's capabilities because of the 640-K memory restriction imposed by MS-DOS. A Markov chain is restricted to 500 or fewer states (without truncation). The 500-state maximum limit is configured for a 512-K memory complement, which was the memory size of the PC AT that was used during PC-HARP development. A 640-K complement will allow a larger model, but the user must adjust various array sizes in the source code and recompile as specified in the HARP user's guide (ref. 6). FEHM's are allowed and single-point failure probabilities are computed, but no near-coincident fault-failure probabilities can be

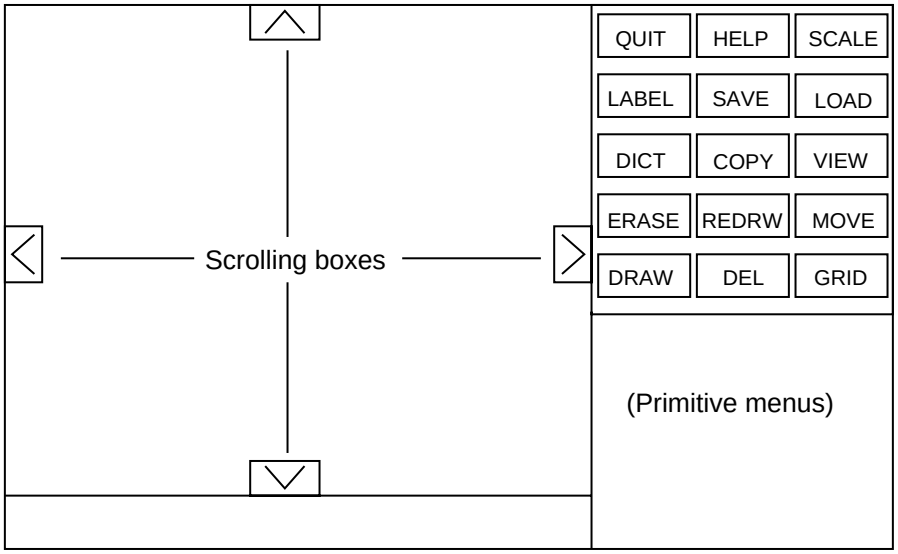

Figure 3. Screen layout.

obtained. Also, PC-HARP does not produce bounds or allow Weibull failure distributions.

Description of Graphics Input Capability

Graphics have been incorporated into the graphics oriented (GO) module of HARP; these graphics offer the user the option of graphical input (ref. 6). The GO module offers a means of inputting a reliability model as a fault tree or Markov chain by using a graphical input language. The GO module converts the user graphical input into files that are read by the next HARP module, TDRIVE for fault trees, FIFACE for a Markov chain. The graphical files are portable, so that files created on a PC can be used with the VAX/VMS version or the Sun/Unix version of HARP. A user can thus create a model with the GO module and solve it on a PC or, if it is too large, transfer the files to a VAX or Sun and execute the VAX or Sun unscaled HARP with these files. The GO program was developed on an IBM PC AT computing platform running under MS-DOS. The source code is written in ANSI standard C. The initial design and implementation of the GO program was done at Duke University. Final design and implementation was completed at Langley Research Center in collaboration with Duke. The GO module is available for use on Sun and VAX workstations and for PC 286/386/486 compatible machines.

The GO module provides static icon menus, a drawing area, and a message area on the screen (fig. 3). The menus are located on the right side of the screen, the drawing area is the large square on the left of the screen, and the message area is the small rectangle below the drawing area. The function menu appears at the upper right of the screen as illustrated in figure 3. For details, see the graphics user guide (ref. 7). Using a mouse, the user makes selections from the function menu or one of the primitive menus. For instance, if a user desires to draw a fault-tree basic event, the user first selects the DRAW icon from the function menu. If the desired icon to be drawn is a circle, the user selects the circle icon from the primitive menu shown in figure 4, and then uses the mouse to position the circle. The message area displays error messages and instructions and provides prompts to the user for which a response from the keyboard may be indicated.

By selecting the appropriate functions, a user can interactively manipulate a fault tree or Markov chain model, exercise file handling, or receive on-line help. Fault trees and Markov chains are constructed by drawing and combining primitives properly. The delete function DEL is used to erase individual primitives. Other functions that allow a user to change a model on the screen are COPY, MOVE, ERASE, and REDRW. The function icons VIEW, SCALE, and HELP facilitate the modeling procedure. File handling is accomplished by the icons LOAD and SAVE.

The dictionary DICT icon enables the creation of a dictionary file, which is required for other HARP modules. During this phase, an interactive interface generates prompts for component names, failure rates, and component FEHM's. If the user chooses to describe an FEHM, the GO program automatically draws certain fault/error-handling models on

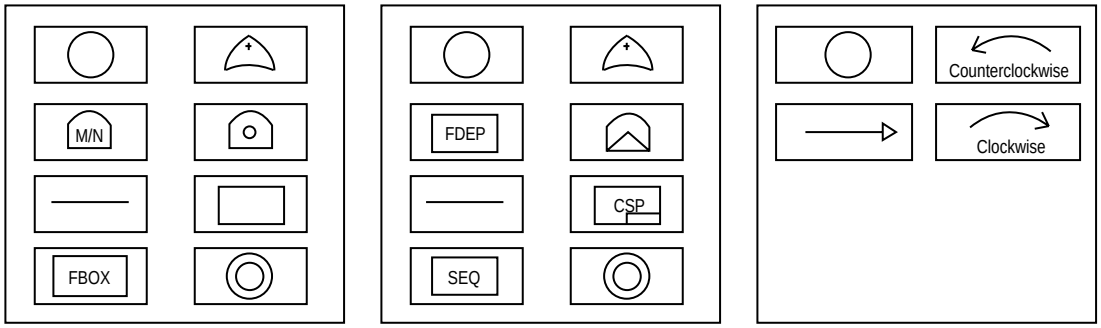

Figure 4. Primitive menus.

the screen as a prompt and continues the interactive interface for an FEHM description. The LABEL icon provides a means of relating the dictionary component types with the model drawn on the screen. During labeling, the GO program leads the user through all the basic events or states by marking each event or state sequentially with a red X. With the aid of the keyboard, the user enters the appropriate component type number as defined in the dictionary. The result of this process is that all fault-tree basic events and any appropriate gates are labeled, or, if the FORM is a Markov chain, all states and transition arcs are labeled.

To further aid the user, there is on-line context-sensitive help available by selection of the HELP function, which provides a brief description of each of the functions in the function menu. There is also a SCALE function that allows the scaling factor of a model to be manipulated. On the sides, top, and bottom of the drawing area are scrolling boxes (fig. 3) with arrows that allow the user to scroll the drawing-area window in four different directions. This feature permits models to be larger than one drawing area. The functionality of the VIEW icon is to display other portions of a model that are not in the immediate drawing-area window. This function causes a small red box to appear in the lower right corner of the drawing area. The box contains a scaled-down representation of the model.

The SAVE and LOAD functions are used to save a model to a file and to retrieve it later for modifications. This feature gives the user the capability of reviewing and editing files at any time during a session or ending a session at any time by simply saving a drawing and quitting.

There are three available primitive menus that are selectable by the user. Figure 4 depicts the choices that appear one at a time on the lower right of the screen. The GO program is executed by using the command "go f" or "go m." By typing "go f," the fault-tree option is selected and the leftmost fault-tree menu in figure 4 becomes visible. The adjacent fault-tree menu is selected by using the right mouse button to toggle between the two fault-tree menus. The command "go m" invokes the Markov chain menu, which will not alternate with the others with the mouse button.

The fault-tree symbols constitute a dynamic fault-tree notation that allows models to be more general than typical combinatorial models. The standard logical AND and OR gates are represented (fig. 4), and there is a circle to symbolize basic events. Also included are an M/N gate, a double circle to represent shared basic events, and an FBOX symbol to mark the top of a fault tree and represent system failure. Four of the symbols represent nonstandard gates that allow sequence and functional dependencies. These symbols include the functional-dependency gate, "FDEP"; the cold-spare gate, "CSP"; the sequence-enforcing gate, "SEQ"; and the priority AND gate, which appears just above the "CSP" gate in figure 4. With these symbols, a user can model dynamic characteristics of a system that could not otherwise be modeled with a combinatorial fault tree (ref. 5).

The order of the input events to the sequence gates is very important. The converter from the fault tree to a Markov chain in the TDRIVE module expects this order to represent the sequence that the reliability modeler intended for the utilization of components. Likewise, the GO module expects the user to draw these inputs in a sequential order from left to right. During the drawing phase, many changes can occur that result in the repositioning of events. The user should be cognizant of the fact that the incoming basic events (circles) to the

sequence gates, particularly the cold-spare gate and the sequence-enforcing gate, are ordered according to their left-to-right position on the screen and not according to the position of the incoming arcs that connect the basic events to the gate.

The double-circle icon in the fault-tree primitive menu is a special symbol that can be used to simplify fault-tree construction. It represents shared (repeated) basic events; that is, the events are not distinct. Normally, this type of basic event has more than one outgoing arc. For the purposes of simplification and readability, GO employs a double circle to represent the sharing of a basic event. If using this symbol in conjunction with the cold-spare gate and the sequence-enforcing gate, the user must realize that the order of the incoming events is reconciled according to the position of the double circle on the screen, and not the position of the event that is shared.

A user completes construction of a fault-tree model when the model (FORM) has been drawn on the screen and saved to a file, the dictionary with any necessary FEHM's is created, and the model is labeled and again saved. The GO program creates files with the appropriate extensions and copies the information from memory to the file in the current directory on a disk. These files include the dictionary file with a ".dic" extension and two fault-tree files with a ".tre" and an ".ftr" extension. If the user specified any FEHM's during the dictionary phase, these files also exist. Upon completion of modeling a Markov chain, a file with an ".mkv" extension is created. For a Markov chain, the user need not explicitly define a dictionary unless coverage is included in the model (ref. 6).

Functionality of Dependency Gates

The last refinements to the GO module include four special fault-tree gates that allow sequence and functional dependencies. These powerful extensions to the GO module and the HARP program were developed jointly by the HARP team members at Duke University and Langley Research Center.

The first new gate, the functional-dependency gate (fig. 5), appeared in HARP version 5 and was developed and used at Duke to describe the integrated airframe propulsion system (IAPSA) reliability model (ref. 8). This gate is the logical equivalent of a combinatorial fault tree composed of AND and OR gates when no fault handling is specified. The action of the functional-dependency gate at first appears strange, since, in most of its applications, it appears to be disconnected from the fault tree. Conceptually, though, its functionality is straightforward

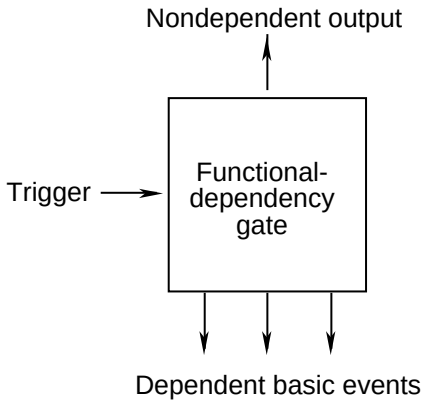

Figure 5. Functional-dependency gate.

and very useful. The input, labeled trigger, can be the output from any gate, whereas the outputs take two forms. The nondependent output simply mimics the trigger input and may or may not be connected to the input of any gate; that is, it can dangle if desired. The typical use of this gate involves the other outputs. The outputs labeled "dependent basic events" must be basic events. Although they are labeled dependent basic events, the basic events themselves are independent. The dependency is related to the trigger event. A typical use of this gate is to account for the functional loss of devices because some other device failed and was therefore unable to provide signal or power input to the downstream operational devices. The notational simplicity of the functional-dependency gate can be better appreciated by viewing the IAPSA fault-tree model discussed in reference 9 or the fault-tolerant parallel processor (FTPP) fault-tree model discussed in reference 5. Although the functionality of this gate has been recognized and reported in the literature (ref. 10), a single gate implementing the functionality has not, to the knowledge of the authors, been reported.

The next gate that was added is a noncombinatorial gate that implements a cold-spare model. This sequence-dependency gate (fig. 6) is naturally called the cold-spare (CSP) gate. In figure 6, the gate output fires (produces an output) when and only when the primary event occurs first, followed by events 1st, 2nd, ...nth. Events 1st, 2nd, ...nth cannot occur first. The primary event can thus represent an active unit, and event 1st is the cold spare that exhibits a zero failure rate until the active unit fails. At that instant, the cold spare is powered up and immediately exhibits a failure rate greater than zero. If additional cold-spare units are added, they are powered up from left to right, and all inputs are independent (possibly

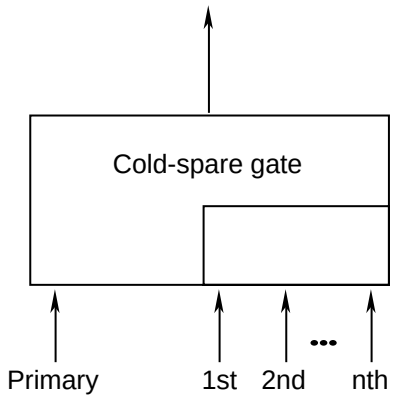

Figure 6. Cold-spare gate.

replicated) basic events. A gate with similar functionality has been reported in the literature (ref. 11) as an inhibit gate. A useful variation of the CSP gate is called the sequence-enforcing gate. This gate is functionally equivalent to the CSP gate, with the exception of how it treats shared events. The inputs of the dependency gate can be (possibly replicated) basic events or the output of some other gate for the primary input only. As with the CSP gate, the sequencing of events is left to right. One important application for this gate is to enable the user to specify a particular fault/error-handling model (FEHM) for a particular state transition entirely as a fault-tree entry. Without this gate, the user is required to initially ignore the placement of the FEHM in the fault tree until its equivalent Markov chain is generated, at which time the notation is inserted into the chain description by using a text editor. This latter scheme was used to model the advanced reconfigurable computer system (ARCS) reported in reference 12. Figures 7 and 8 show the ARCS fault trees without and with the sequence-enforcing gate.

Figure 7 depicts the hydraulics stage as "REP 3 of (h)," which means the hydraulics stage includes three devices with identical failure rates. This HARP notation signals the conversion routine from fault tree to Markov chain to aggregate Markovian states; the model state size is thus reduced. With this notation, HARP assigns the same FEHM to all three devices, and this FEHM is placed between all Markovian transitions except the one prior to system failure. In the ARCS model, fault handling is assumed to be perfect when only one fault occurs in one of three voting devices; therefore, it is undesirable to have the same FEHM assigned to all state transitions. The sequence-enforcing gate offers a solution that can be applied at the fault-tree level.

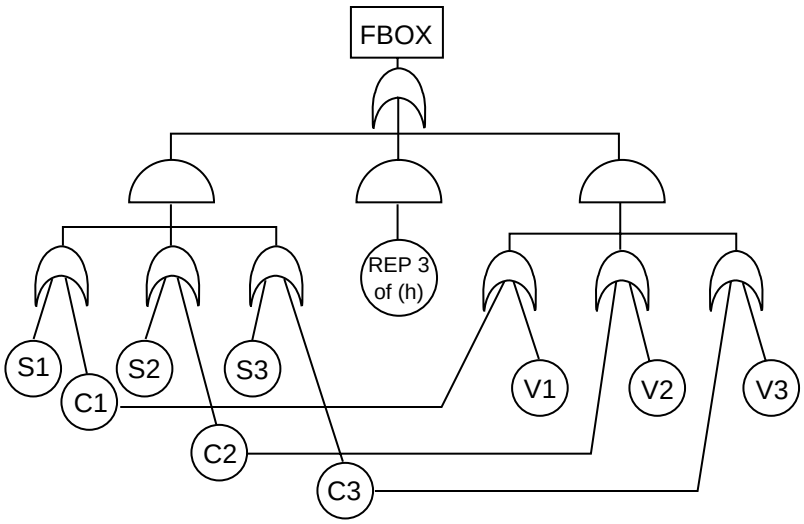

Figure 7. Fault-tree representation of ARCS.

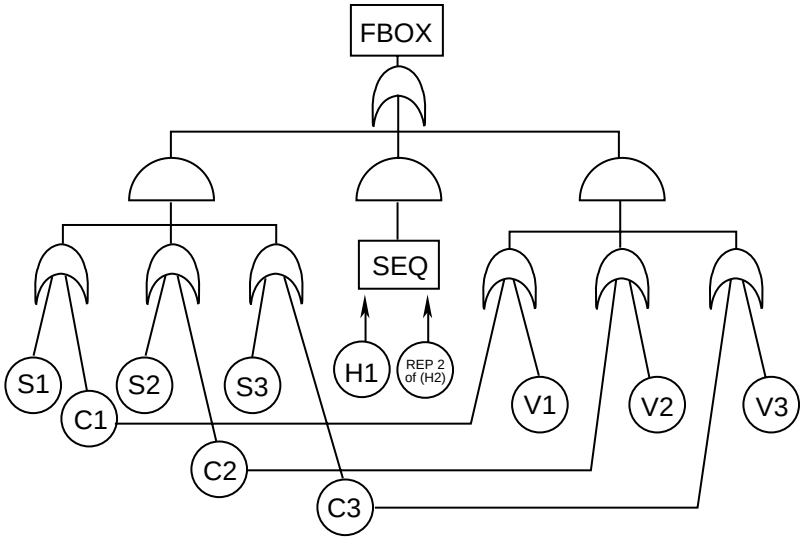

Figure 8. State-dependent FEHM for hydraulics.

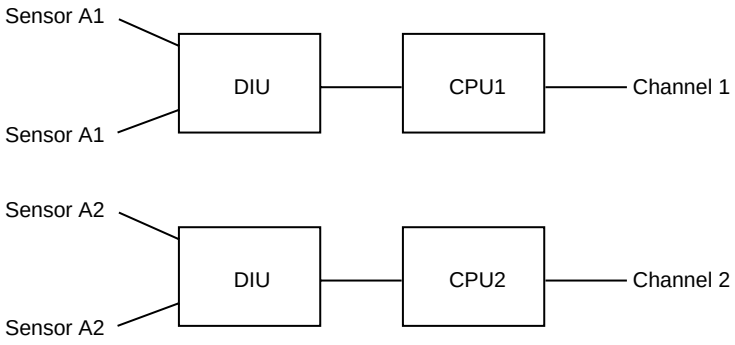

Figure 9. Two-channel system.

Figure 8 shows the sequence-enforcing gate with inputs "H1" and "REP 2 of (H2)." By splitting up the replication of three hydraulics to one nonreplicated "H1" device and two replicated "H2" devices, no FEHM can be assigned to "H1" as desired (i.e., assigns perfect fault handling), but any single FEHM can be assigned to the other two devices, "H2." All three devices could still have the same failure rate, of course. A complete application of the sequence-enforcing gate for ARCS for all devices, and not just the hydraulics as illustrated above, is necessary to accurately replicate the results of reference 3. The cold-spare gate and the sequence-enforcing gate differ primarily in the way they treat shared events (ref. 6).

The last new addition of sequence-dependency gates is called the priority AND (P-AND) gate. The P-AND gate differs from a combinatorial AND gate in only one respect—in HARP only two inputs for the P-AND are allowed, and the gate produces an output only if the leftmost event occurs first and is followed by the rightmost event. Contrary to the CSP gate, the rightmost event in a P-AND can occur first, but no output results. The functionality, the name of the gate, and the gate symbol were directly obtained from reference 13. The HARP team members recognized the need for this functionality for application to fault-tolerant systems before the priority AND gate was recognized in the literature by the team. The particular system that prompted our interest is shown in figures 9 and 10—a two-channel system (ref. 6).

Our experience with the use of these new gate additions to HARP has been somewhat extensive. We have applied them to some very complicated fault-tolerant network systems (ref. 5). Although there is no warm-spare gate, that model has been functionally emulated with the existing gates (ref. 14). Pooled spares models have also been emulated (ref. 15). With the HARP Markov chain truncation technique that bounds the truncation error, extremely large Markov chains have been modeled and solved that have simple fault-tree diagrams. These

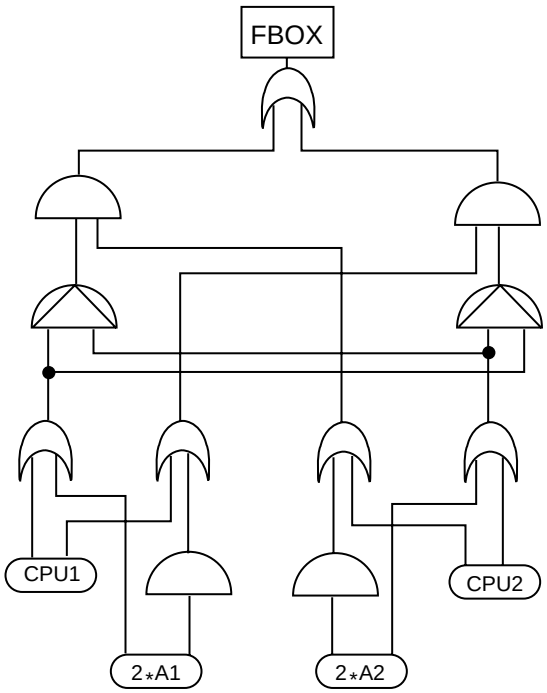

Figure 10. Fault-tree model of two-channel system.

models have demonstrated the succinct yet powerful notational value of the HARP extended fault-tree capability.

Experiences and Results

The GO module was originally designed to operate on an IBM PC AT with a color monitor using MS-DOS, but has since been modified to execute in both Sun and VAX/VMS workstation environments. The designers of GO selected the C programming language paired with the graphical kernel system (GKS) graphics standard for program development. Language C was chosen because of its flexibility and robustness, and GKS was selected based on its reputation as a device-independent graphics standard (ref. 2); thus, there are ideally portable graphics applications between the systems. The software and resources needed to execute the GO module of HARP vary depending on the workstation. All versions of the executable code require a minimum of 300 K bytes of memory, GKS device drivers, header files, and several static FEHM files. In addition, the PC requires a minimum of 512 K bytes of memory, an EGA/VGA graphics display board, a math coprocessor, and a mouse. The program relies on the graphic software system (GSS) GKS for the PC, SunGKS for the Sun, and VAX GKS for the VAX. Each vendor provides a C language binding. The purpose of the binding is to standardize the GKS function names and the number and order of function parameters, which permits programs written with a specific language binding to be used with any GKS implementation.

In converting the GO module from GSS*GKS to SunGKS, many dissimilarities were found between the two implementations of GKS. Most of the GSS function calls had to be reconstructed because of differences found in function-naming conventions, inconsistent parameter passing, and deviating definitions for GKS types. The differences were mainly attributed to the GSS decision to implement the C language binding to resemble a Fortran binding. Although both SunGKS and VAX GKS adhere to the proposed ANSI C language binding standard, three salient differences exist between them. The major distinction between the two implementations relates to the amount of initialization each function needs before calling. For example, when calling a SunGKS function, the programmer only needs to declare the function or parameter as a specific GKS C type definition, and all memory needed to perform the function is allocated. To accomplish the same objective using the VAX GKS C binding, the programmer is responsible for designating all memory necessary for that function. Another area of deviation concerns workstation specifics. For example, when using SunGKS, certain features, such as changing the default size of a GKS window, are tied to utilities written for Suntools, a windowing environment for the Sun. Since these utilities are specifically written to work only in the Suntools environment, the application is not portable to other systems. Other variations concern differing syntax for enumerated and GKS type definitions and differences between workstation attributes, such as indexing colors, markers, prompts, and echo types. Because of the lack of standardization of the C language binding and because each vendor incorporates machine dependencies into their implementation, developing portable applications is very difficult.

Concluding Remarks

The result of this effort is an engineering workstation environment that is attainable on several different platforms and offers a high-level graphical input capability for reliability and availability modeling. The fault-tree notation is an important and powerful modeling feature that simplifies model input. The dynamic sequence-dependency gates bring the flexibility and modeling power of Markov chains to fault-tree users without the accompanying, and often confusing, modeling detail typical of Markov chains.

Because the graphics oriented (GO) module supplements the other hybrid automated reliability predictor (HARP) modules, the user has the option of choosing to input information either textually or graphically without being concerned about file formats or portability. The GO module offers the user the advantage of a familiar notation expressed as a visual image on a computer screen or a printable image on a plotter by using the appropriate graphical kernel system (GKS) device drivers.

Acknowledgments

Original design and code implementation of the GO module was done by Elizabeth Rothmann. The original design of HARP was done by Kishor S. Trivedi, Joanne Bechta Dugan, Robert M. Geist, and Mark K. Smotherman. The original HARP implementors are Elizabeth Rothmann, Joanne Bechta Dugan, Mark Boyd, Mark K. Smotherman, and Nitin Mittal.

NASA Langley Research Center
Hampton, VA 23665-5225
October 31, 1991

References

1. Dugan, Joanne Bechta; Trivedi, Kishor S.; Smotherman, Mark K.; and Geist, Robert M.: The Hybrid Automated Reliability Predictor. *AIAA J. Guid., Control & Dyn.,* vol. 9, no. 3, May–June 1986, pp. 319–331.

2. McKay, Lucia: *GKS Primer.* Nova Graphics International Corp., 1984.

3. Bavuso, Salvatore J.; Dugan, Joanne Bechta; Trivedi, Kishor; Rothmann, Beth; and Boyd, Mark: *Applications of the Hybrid Automated Reliability Predictor—Revised Edition.* NASA TP-2760, 1988.

4. Geist, Robert; Smotherman, Mark; Trivedi, Kishor; and Dugan, Joanne Bechta: The Reliability of Life-Critical Computer Systems. *Acta Inform.,* vol. 23, 1986, pp. 621–642.

5. Dugan, Joanne Bechta; Bavuso, Salvatore J.; and Boyd, Mark A.: Fault Trees and Sequence Dependencies. *Annual Reliability and Maintainability Symposium—1990 Proceedings,* IEEE Catalog No. 90CH2804-3, Inst. of Electrical and Electronics Engineers, Inc., 1990, pp. 286–293.

6. Rothmann, Elizabeth; Dugan, Joanne Bechta; Trivedi, Kishor S.; Boyd, Mark; Mittal, Nitin; and Bavuso, Salvatore J.: *HARP: The Hybrid Automated Reliability Predictor—Introduction and Guide for Users, HARP Version 6.1.* Dep. of Computer Science, Duke Univ., May 1990.

7. Rothmann, Elizabeth; Mittal, Nitin; Howell, Sandra; and Bavuso, Salvatore J.: *HARP: The Hybrid Automated Reliability Predictor—Guide for Graphics Users, HARP Version 6.1.* Dep. of Computer Science, Duke Univ., Oct. 1990.

8. Cohen, G. C.; Lee, C. W.; Brock, L. D.; and Allen, J. G.: *Design Validation Concept for an Integrated Airframe Propulsion Control System Architecture (IAPSA II).* NASA CR-178084, 1986.

9. Veeraraghavan, Malathi: Modeling and Evaluation of Fault-Tolerant Multiple Processor Systems. Ph.D. Thesis, Duke Univ., 1988.

10. Martensen, Anna L.; and Bavuso, Salvatore J.: *Tutorial and Hands-On Demonstration of a Fluent Interpreter for CARE III.* NASA TM-4011, 1987.

11. Chatterjee, Purnendu: *Fault Tree Analysis: Reliability Theory and Systems Safety Analysis.* ORC 74-34 (Contracts N00014-69-A-0200-1036 and F33615-73-C-4078), Univ. of California, Nov. 1974. (Available from DTIC as AD A004 209.)

12. Bjurman, B. E.; Jenkins, G. M.; Masreliez, C. J.; McClellan, K. L.; and Templeman, J. E.: *Airborne Advanced Reconfigurable Computer System (ARCS).* NASA CR-145024, 1976.

13. Fussell, J. B.; Aber, E. F.; and Rahl, R. G.: On the Quantitative Analysis of Priority-AND Failure Logic. *IEEE Trans. Reliab.,* vol. R-25, no. 5, Dec. 1976, pp. 324–326.

14. Boyd, Mark; and Tuazon, Jesus O.: Fault Tree Models for Fault Tolerant Hypercube Multiprocessors. *Annual Reliability and Maintainability Symposium—1991 Proceedings,* IEEE Catalog No. 91CH2966-0, Inst. of Electrical and Electronics Engineers, Inc., 1991, pp. 610–614.

15. Dugan, Joanne Bechta; Bavuso, S. J.; and Boyd, M. A.: Modeling Advanced Fault-Tolerant Systems With HARP. *Topics in Reliability & Maintainability & Statistics, Consolidated Lecture Notes—1991 "Tutorial Notes," Annual Reliability and Maintainability Symposium,* 1991, pp. FTS-i–FTS-25.